Computer Organization, Spring, 2015

Lab 4: Pipelined CPU

Due: 2015/6/4 11:59pm

Demo: to be announced

1. Goal:

To modify the CPU designed in lab3 and implement a simple version pipelined CPU.

2. HW requirement:

a. Please use Xilinx as simulation platform.

b. Please attach your name and student ID as comments at the top of each file.
PLEASE FOLLOW THE FOLLOWING RULE! Zip your folder and name it as "ID.zip" (e.g., 0216001.zip) before uploading to e3. Multiple submissions are accepted, and the version with the latest time stamp will be graded.

Note: You must be uploading the ".zip" file to e3.

c. Pipe_Reg.v, Reg_File.v and Pipe_CPU_1.are supplied.

d. Must use the supplied Reg_File.v

e. In the top module, based on your design, please accordingly change the following *N* to the value which is the total size (length) of input signals (including data and control) entering each set of pipeline registers.

Pipe_Reg #(.size(N)) ID_EX

f. And notice that, when using CO_P4_test_data1 and CO_P4_data2, these lines in Data_Memory.v should be commented (shown in the following figure). However, these will be used for CO_P4_test_data3.

```
initial begin
    for(i=0; i<128; i=i+1)
        Mem[i] = 8'b0;
    /*Mem[0] = 8'b0100;
    Mem[4] = 8'b0101;
    Mem[8] = 8'b0110;
    Mem[12] = 8'b0111;
    Mem[16] = 8'b1000;
    Mem[20] = 8'b1001;
    Mem[24] = 8'b1010;
    Mem[28] = 8'b0010;
    Mem[32] = 8'b0001;
    Mem[36] = 8'b0011;*/
end
```

3. Requirement description:

a. Code (120%):

Basic instruction set (80%): Previous Labs' instructions, such as ADD, ADDI, SUB, AND, OR, SLT, SLTI, SRL, SRLV, LW, SW, BEQ, and MUL. And there is no JUMP in this Lab.

Advance (40%): Hazard Detection + Forwarding

Forward data if instructions have data dependency.

Stall pipelined CPU if it detects load-use.

Flush if control/branch hazards occur.

b. Testbench:

CO_P4_test_1.txt for testing basic instructions, CO_P4_test_2.txt for testing forwarding and stalling, CO_P4_test_3.txt for testing all features including flushing.

CO_P4_test_1.txt

```
Begin:
addi    $1, $0, 3;      // a = 3
addi    $2, $0, 4;      // b = 4
addi    $3, $0, 1;      // c = 1
sw      $1, 4($0);      // A[1] = 3
add     $4, $1, $1;     // $4 = 2a
or      $6, $1, $2;     // e = a | b
and     $7, $1, $3;     // f = a & c
sub     $5, $4, $2;     // d = 2a - b
slt     $8, $1, $2;     // g = a < b
beq     $1, $2, Begin; // if b==h goto Begin
lw      $10, 4($0);     // i = A[1]
```

CO_P4_test_2.txt

```
addi    $1, $0, 16
addi    $2, $1, 4
addi    $3, $0, 8
sw      $1, 4($0)
lw      $4, 4($0)
sub     $5, $4, $3
```

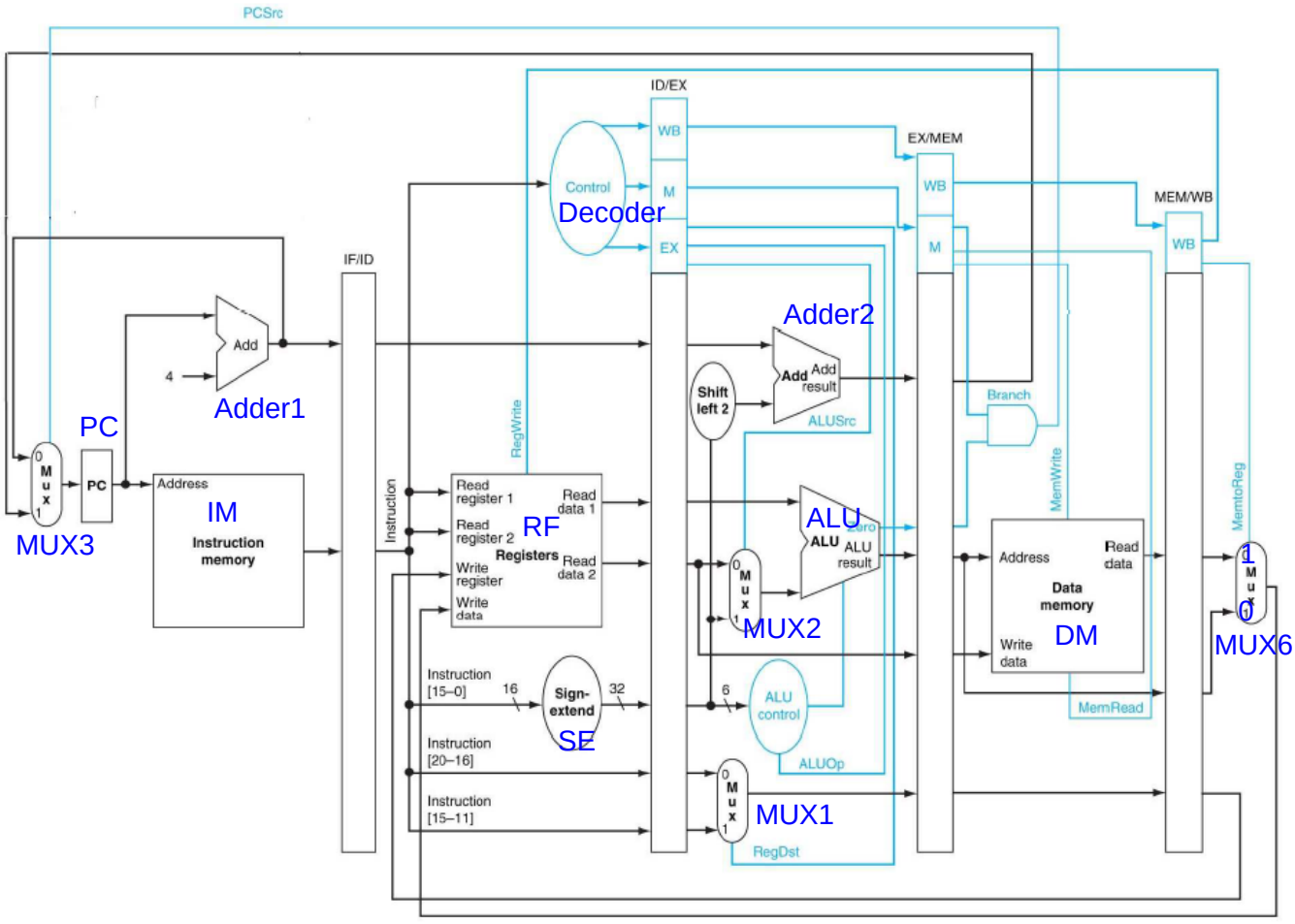
```
add     $6, $3, $1
addi    $7, $1, 10
and     $8, $7, $3
addi    $9, $0, 100
```

c. Report (20%):

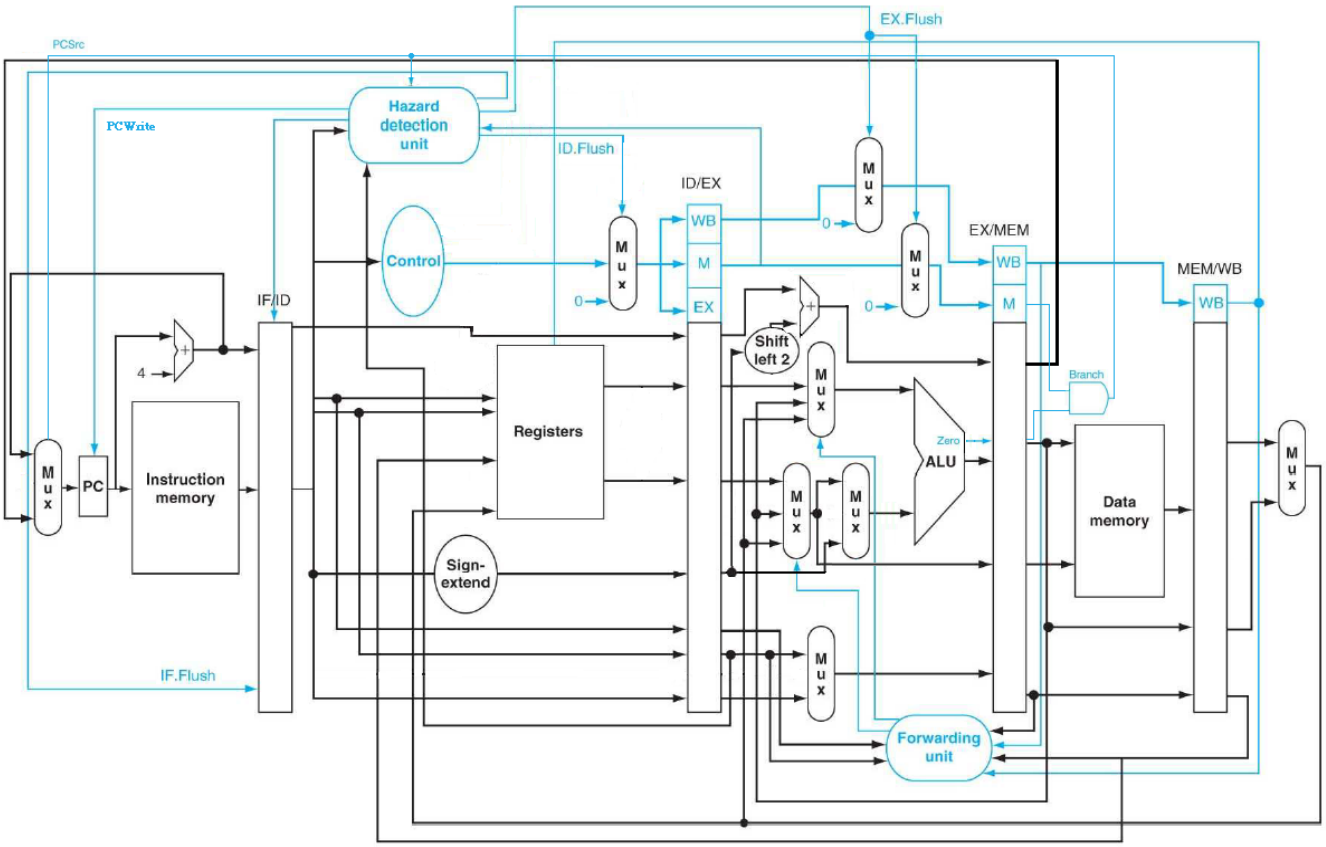
The context must include:

1. Source code and the note
2. Your architecture
3. Hardware module analysis
4. Finished part
5. Problems you met and solutions
6. Summary

4. Architecture (basic):



5. Architecture (advance):



6. Grade

a. Total score: 140% COPY WILL GET 0!!
b. Basic score: 80%
c. Advance score: 40%
d. Report: 20%
e. Delay: 10% off / day

7. Hand in your Assignment

Please upload the assignment to the E3.
Put all of .v source files and report into same compressed file. (Use your student ID to be the name of your compressed file and must have the form of “student ID”. Ex. 0216001.zip)

8. Demo

Time: to be announced

Please review your code from lab 1 to lab 4. All of the questions will be related to the labs. You will get a grade after the demo, and it will be part of your lab score.

9. Q&A

If you have any questions, use E3 discussion forum or just send e-mail to TA.